WHAT IS CLAIMED IS:

1. A method for inserting new instructions between instruction items of preexisting object code, at least some of said instruction items containing addresses, said method comprising the steps of:

a) providing a new object code table for storing items in;

b) storing, for each instruction item in the preexisting object code, a new code block into the new object code table, said new code block comprising any desired new instructions and the instruction item; and

c) updating addresses in instruction items in the new object code table.

2. The method of claim 1, further comprising after the updating step the step of creating a relocatable object code file using the updated new object code table.

3. The method of claim 1, further comprising the step of

a) providing an inter-item offset table and a forward table, wherein tables are for storing items in, said items having locations within tables; and wherein

b) the storing step further comprises performing the following steps for each instruction item in the preexisting object code:

i) storing into the forward table a location for the item within the new object code table, said location being a location within the new code block; and

ii) for items which contain an inter-item offset, indicating the offset in the inter-item offset table; and

c) the updating step comprises, for each inter-item offset indicated in the inter-item offset table, updating the inter-item offset using the forward table.

4. The method of claim 3, wherein

a) the step of providing the forward table comprises providing a forward table which includes a forward index table and a forward control index table;

b) the location storing step comprises

i) storing the location of the item within the new object code table into the forward index table;

ii) storing the location of the new code block within the new object code table into the forward control index table; and

c) the updating step further comprises using the forward control index table for inter-item offsets which are for program control transfer, and using the forward index table otherwise.

5. A method for inserting new instructions into preexisting instructions, said method comprising the steps of:

a) providing an old object code table containing the preexisting instructions;

b) providing a new object code table, an inter-item offset table, a forward index table, and a forward control index table, said tables being for storing items in, said items having locations within said tables;

c) performing the following steps for each item in the old object code table:

i) storing a new code block into the new object code table, said new code block comprising any desired new instructions and the item;

ii) storing the location of the item within the new object code table into the forward index table;

iii) storing the location of the new code block within the new object code table into the forward control index table; and

iv) for items which contain an inter-item offset, indicating the offset in the inter-item offset table; and

d) for each inter-item offset indicated in the inter-item offset table, updating the inter-item offset, using the

for program control transfer, and using the forward index table otherwise.

6. The method of claim 5, wherein

a) the offset indicating step comprises, for items containing an inter-item offset referenced from a base and pointing to a target, indicating in the inter-item offset table the locations of the target and base within the old object code table and of the offset in the new object code table; and

b) the updating step comprises, for offsets indicated in the inter-item offset table, determining a forward location for the base using the forward index table, determining a forward location for the target using the forward control index table for targets which are program control transfer destinations and using the forward index table otherwise, and updating the inter-item offset in the new object code table using the forward locations of the base and target.

7. A method for inserting new instructions into a relocatable object file having preexisting instructions, data, and linking and debugging auxiliary structures having at least some offsets associated with the preexisting instructions and data, said method comprising the steps of:

a) providing an old object code table, a new object code table, an inter-item offset table, a forward index table, and a forward control index table, said tables being for storing items in, said items having locations within said tables;

b) copying the preexisting instructions and data of the object file into the old object code table;

c) performing the following steps for each item in the old object code table:

i) storing a new code block into the new object code table, said new code block comprising any desired new instructions and the item;

ii) storing the location of the item within the new object code table into the forward index table;

iii) storing the location of the new code block within the new object code table into the forward control inde: table; and

iv) for items which contain an inter-item offset pointing to a target and referenced from a base, indicating in the inter-item offset table the locations of the target and base within the old object code table and of the offset in the new object code table, and modifying the item, if necessary, t be consistent with a maximum-byte offset;

d) for each inter-item offset indicated in the inter-item offset table, determining a forward location for the base using the forward index table, determining a forward location for the target using the forward control index table for targets which are program control transfer destinations and using the forward index table otherwise, and updating the inter-item offset in the new object code table using the forward locations of the base and target;

e) updating the offsets in the auxiliary structures; and

f) creating a new relocatable object file comprising the new object code table and the updated auxiliary structures

8. The method of claim 7, wherein the auxiliary structures include text relocation structures and data relocation structures, and wherein the step of updating offsets in auxiliary structures comprises updating offsets in the text and data relocation structures using the forward index table.

9. The method of claim 8, wherein said object file also has symbol structures, said method further comprising the step of

for symbol structures that contain an instruction offset, updating the instruction offset using the forward control index table.

10. A method for inserting new instructions into a relocatable object file having preexisting instructions, data, and auxiliary structures, said auxiliary structures having at

data relocation structures, and symbol structures, said method comprising the steps of:

a) providing an entry point table, a new object code table, an inter-item offset table, a forward index table, and a forward control index table, said tables being for storing items in, said items having locations within said tables;

b) providing an old object code table containing the preexisting instructions and data of the object file;

c) for each symbol structure associated with a function and having a function entry address, storing the function entry address in the entry point table, said entry address indicating in the old object code table a first instruction of the function, the function also having a last instruction, all items between the first instruction and last instruction also being instructions of the function;

d) performing the following steps for each entry address stored in the entry point table:

i) performing the following steps for each preexisting instruction of the function indicated by the entry address:

1) storing a new code block into the new object code table, said new code block comprising any desired new instructions and the preexisting instruction;

2) storing the location of the preexisting instruction within the new object code table into the forward index table;

3) storing the location of the new code block within the new object code table into the forward control index table; and

4) for preexisting instructions that contain an inter-item offset pointing to a target and referenced from a base, indicating in the inter-item offset table the locations of the target and base within the old object code table and of the offset in the new object code table, and modifying the preexisting instruction in the new code block, if necessary, to be consistent with a maximum-byte offset; and

ii) performing the following steps for each item in

instruction of the function indicated by the entry address and before a closest next entry address:

1) copying the item into the new object code table;

2) storing the location of the item within the new object code table into the forward index table;

e) for each inter-item offset indicated in the inter-item offset table, determining a forward location for the base using the forward index table, determining a forward location for the target using the forward control index table for targets which are program control transfer destinations and using the forward index table otherwise, and updating the inter-item offset in the new object code table using the forward locations of the base and target;

f) updating offsets in the auxiliary structures; and

g) creating a new relocatable object file comprising the new object code table and the updated auxiliary structures

11. A method for signalling errors in substantially all memory accesses by a computer program, said computer program capable of accessing memory locations within a set of memory regions, said method comprising the steps of:

a) maintaining memory access status information for substantially all memory locations in the set of accessible memory regions; and

b) for substantially all memory accesses by the computer program, performing the following steps:

i) determining an access type of the memory access; and

ii) signalling an error if the type of the memory access is inconsistent with the memory access status information for the memory location accessed.

12. The method of claim 11, wh rein the maintaining step comprises the step of recording the memory access status information as status codes in a dedicated table having entries corresponding to memory addresses.

13. The method of claim 11, wherein memory access types include read and write, and wherein:

a) the maintaining step comprises maintaining status information indicating two states of dynamically allocated memory, said states being allocated and unallocated; and

b) the signalling step comprises

i) signalling an error if a memory location having unallocated status is accessed by a memory access of a write type; and

ii) signalling an error if a memory location having unallocated status is accessed by a memory access of a read type.

14. The method of claim 13, wherein the maintaining step comprises indicating that a memory region is unallocated by storing a predetermined bit pattern in the memory region, and indicating that a memory region is allocated by storing any of a predetermined set of bit patterns in the memory region.

15. The method of claim 13, wherein the maintaining step further comprises the step of recording the memory access status information as status codes in a dedicated table having entries indexed by memory address.

16. The method of claim 11, wherein memory access types include read and write, and wherein:

a) the maintaining step comprises maintaining status information indicating three states of dynamically allocat d memory, said states being allocated-and-initialized, allocated-and-uninitialized, and unallocated; and

b) the signalling step comprises

i) signalling an error if a memory location having unallocated status is accessed by a memory access of a write type;

ii) signalling an error if a memory location having unallocated status is accessed by a memory access of a read

iii) signalling an error if a memory location having allocated-and-uninitialized status is accessed by a memory access of a read type.

17. The method of claim 16, wherein the maintaining step comprises:

i) indicating that a memory region is unallocated by storing a first predetermined bit pattern in the memory region;

ii) indicating that a memory region is allocated-and-uninitialized by storing a second predetermined bit pattern in the memory region; and

iii) indicating that a memory region is allocated by storing any other bit pattern in the memory region.

18. The method of claim 16, wherein the maintaining step further comprises the step of recording the memory access status information as status codes in a dedicated table having entries corresponding to memory addresses.

19. A method of equipping a computer program with the ability to monitor most of its own memory accesses, said computer program having preexisting code items, said method comprising the steps of

a) for most preexisting code items the performance of which involves a memory read, adding memory read monitoring code located so that it will be executed each time the memory read preexisting code item is executed; and

b) for most preexisting code items the performance of which involves a memory write, adding memory write monitoring code located so that it will be executed each time the memory write preexisting code item is executed.

20. The method of claim 19, wherein th computer program has source files, and wherein preexisting code items the performance of which involves a memory read or a memory write are identified by parsing the source files of the computer

21. The method of claim 19, wherein the computer program has a structured representation constructed by a compiler, and wherein

a) preexisting code items the performance of which involves a memory read or a memory write are identified by examining the structured representation; and

b) said structured representation is augmented to include the memory read monitoring code and the memory write monitoring code.

22. A method for equipping a computer program with the ability to monitor most of its own memory accesses, wherein memory access types include read and write, said method comprising the steps of:

performing the following steps for substantially all relocatable object files for the program, said relocatable object files containing instruction items of preexisting object code, said instruction items having locations within the preexisting object code:

a) providing a new object code table for storing items in;

b) storing, for each instruction item in the preexisting object code, a new code block into the new object code table, said new code block comprising the item and

i) memory read monitoring code for items the performance of which involves a memory access of a read type; and

ii) memory write monitoring code for items the performance of which involves a memory access of a write type;

c) updating offsets in the new object code table;

d) creating a new relocatable object file comprising the code items from the new object code file.

23. The method of claim 22, wherein said computer program is capable of accessing a set of memory regions, said method further comprising the step of

adding code to establish memory access status information for substantially all memory locations in the set of accessible memory regions, said status information indicating three states of dynamically allocated memory, said states being allocated-and-initialized, allocated-and-uninitialized, and unallocated; and

wherein

the storing step comprises

i) adding allocation indicating code for items the performance of which would involve allocating memory, said code updating the status information for the allocated memory to be in the allocated-and-unitialized state;

ii) adding deallocation indicating code for items the performance of which would involve deallocating memory, said code updating the status information for the allocated memory to be in the unallocated state;

iii) for adding memory read monitoring code, adding code to signal an error if the accessed memory is not in the allocated-and-initialized state;

iv) for adding memory write monitoring code, adding code signalling an error if the accessed memory is in the unallocated state, otherwise updating the status information for the allocated memory to be in the allocated-and-initialized state if the accessed memory is not in the unallocated state.

24. The method of claim 22, wherein

a) the providing step further comprises providing, an inter-item offset table, and a forward table, wherein tabl s are for storing items in, said items having locations within tables;

b) the storing step further comprises performing the following steps for each instruction item in the preexisting object code:

i) storing into the forward table a location for the item within the new object code table, said location being a location within the new code block; and

ii) for items which contain an inter-item offset,

c) the updating step comprises, for each inter-item offset indicated in the inter-item offset table, updating the inter-item offset using the forward table.

25. The method of claim 24, wherein

a) the forward table comprises a forward index table and a forward control index table;

b) the location storing step comprises

i) storing the location of the item within the new object code table into the forward index table;

ii) storing the location of the new code block within the new object code table into the forward control index table; and

c) the updating step comprises using the forward control index table for inter-item offsets which are for program control transfer, and using the forward index table otherwise.

26. In a computer, an executable computer program capable of monitoring most of its own memory accesses, wherein memory access types include read and write, said computer program capable of accessing a set of memory regions, said computer program comprising

a) status information maintenance code for maintaining memory access status information in a dedicated table having entries corresponding to memory addresses for at least most memory locations in the set of accessible memory regions, said status information indicating three states of dynamically allocated memory, said states being allocated-and-initialized, allocated-and-uninitialized, and unallocated;

b) write error signalling code for signalling an error if a memory location having unallocated status is accessed by a memory access of a write type;

c) read error signalling code for signalling an error if a memory location having unallocated status is accessed by a memory access of a read type and if a memory location having allocated-and-uninitialized status is accessed by a memory